AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

Listing of Claims:

1. (Currently amended) A method to automate isolation of native code within a computer program that has been compiled to a platform-independent code, the method comprising:

receiving a library containing a native code sub-routine, wherein the native code sub-routine provides a service to the computer program;

analyzing the library to determine a defined symbol name for the native code sub-routine;

creating a proxy sub-routine for the native code sub-routine, wherein the proxy sub-routine forms a link to the native code sub-routine;

placing the proxy sub-routine into a new library using the defined symbol name of the native code sub-routine as a symbol name for the proxy sub-routine;

running the native code sub-routine in a first process;

executing the platform-independent code in a second process; and

invoking the native code sub-routine in the first process by ~~calling~~ using an interprocess call to call the proxy sub-routine from the platform-independent code in the second process;

wherein operations in the first process are isolated from memory and other system resources belonging to the second process by ensuring that interactions between the first process and the second process take place through the interprocess call, so that an error in the first process does not, one of, corrupt

memory belonging to the second process or interfere with the second process in any other way.

2. (Original) The method of claim 1, further comprising:

providing a proxy platform-independent native interface (PINI) to the library containing the native code sub-routine; and

transparently transforming local PINI calls into calls to the proxy PINI,

wherein transforming local PINI calls into calls to the proxy PINI preserves an original control flow, and

wherein upcalls and downcalls are guaranteed to be executed by a same thread of a process that executes the native code sub-routine.

3. (Original) The method of claim 1, wherein analyzing the library to determine the defined symbol name includes analyzing the library to determine call arguments for the defined symbol name.

4. (Original) The method of claim 3, wherein analyzing the library to determine call arguments for the defined symbol name is accomplished at runtime by analyzing a current call frame.

5. (Original) The method of claim 3, further comprising copying call arguments from the proxy sub-routine to a call to the native code sub-routine.

6. (Original) The method of claim 3, further comprising returning a result value from the native code sub-routine to the proxy sub-routine.

7 (Canceled).

8. (Original) The method of claim 1, wherein the proxy sub-routine and the native code sub-routine communicate through inter-process communication.

9. (Original) The method of claim 1, wherein forming the link to the native code sub-routine includes translating a data element from a first address width in the computer program to a second address width in the native code sub-routine.

10. (Currently amended) A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method to facilitate automated isolation of native code within a computer program that has been compiled to a platform-independent code, the method comprising:

receiving a library containing a native code sub-routine, wherein the native code sub-routine provides a service to the computer program;

analyzing the library to determine a defined symbol name for the native code sub-routine;

creating a proxy sub-routine for the native code sub-routine, wherein the proxy sub-routine forms a link to the native code sub-routine;

placing the proxy sub-routine into a new library using the defined symbol name of the native code sub-routine as a symbol name for the proxy sub-routine;

running the native code sub-routine in a first process;

executing the platform-independent code in a second process; and

invoking the native code sub-routine in the first process by ~~calling~~ using an interprocess call to call the proxy sub-routine from the platform-independent code in the second process;

wherein operations in the first process are isolated from memory and other system resources belonging to the second process by ensuring that interactions between the first process and the second process take place through the interprocess call, so that an error in the first process does not, one of, corrupt

memory belonging to the second process or interfere with the second process in any other way.

11. (Original) The computer-readable storage medium of claim 10, the method further comprising:

providing a proxy platform-independent native interface (PINI) to the library containing the native code sub-routine; and

transparently transforming local PINI calls into calls to the proxy PINI,

wherein transforming local PINI calls into calls to the proxy PINI preserves an original control flow, and

wherein upcalls and downcalls are guaranteed to be executed by a same thread of a process that executes the native code sub-routine.

12. (Original) The computer-readable storage medium of claim 10, wherein analyzing the library to determine the defined symbol name includes analyzing the library to determine call arguments for the defined symbol name.

13. (Original) The computer-readable storage medium of claim 12, wherein analyzing the library to determine call arguments for the defined symbol name is accomplished at runtime by analyzing a current call frame.

14. (Original) The computer-readable storage medium of claim 12, the method further comprising copying call arguments from the proxy sub-routine to a call to the native code sub-routine.

15. (Original) The computer-readable storage medium of claim 12, the method further comprising returning a result value from the native code sub-routine to the proxy sub-routine.

16 (Canceled).

17. (Original) The computer-readable storage medium of claim 10, wherein the proxy sub-routine and the native code sub-routine communicate through inter-process communication.

18. (Original) The computer-readable storage medium of claim 10, wherein forming the link to the native code sub-routine includes translating a data element from a first address width in the computer program to a second address width in the native code sub-routine.

19. (Currently amended) An apparatus that facilitates automated isolation of native code within a computer program that has been compiled to a platform-independent code, the apparatus comprising:

a receiving mechanism that is configured to receive a library containing a native code sub-routine, wherein the native code sub-routine provides a service to the computer program;

an analyzing mechanism that is configured to analyze the library to determine a defined symbol name for the native code sub-routine;

a creating mechanism that is configured to create a proxy sub-routine for the native code sub-routine, wherein the proxy sub-routine forms a link to the native code sub-routine;

a placing mechanism that is configured to place the proxy sub-routine into a new library using the defined symbol name of the native code sub-routine as a symbol name for the proxy sub-routine;

a running mechanism that is configured to run the native code sub-routine in a first process;

an executing mechanism that is configured to execute the platform-independent code in a second process; and

an invoking mechanism that is configured to invoke the native code sub-routine in the first process by ~~calling~~ using an interprocess call to call the proxy sub-routine from the platform-independent code in the second process;

wherein operations in the first process are isolated from memory and other system resources belonging to the second process by ensuring that interactions between the first process and the second process take place through the interprocess call, so that an error in the first process does not, one of, corrupt memory belonging to the second process or interfere with the second process in any other way.

20. (Original) The apparatus of claim 19, further comprising:

a providing mechanism configured to provide a proxy platform-independent native interface (PINI) to the library containing the native code sub-routine; and

a transforming mechanism that is configured to transparently transform local PINI calls into calls to the proxy PINI,

wherein transforming local PINI calls into calls to the proxy PINI preserves an original control flow, and

wherein upcalls and downcalls are guaranteed to be executed by a same thread of a process that executes the native code sub-routine..

21. (Original) The apparatus of claim 19, wherein the analyzing mechanism is further configured to analyze the library to determine call arguments for the defined symbol name.

22. (Original) The apparatus of claim 21, wherein analyzing the library to determine call arguments for the defined symbol name is accomplished at runtime by analyzing a current call frame.

23. (Original) The apparatus of claim 21, further comprising a copying mechanism that is configured to copy call arguments from the proxy sub-routine to a call to the native code sub-routine.

24. (Original) The apparatus of claim 21, further comprising a returning mechanism that is configured to return a result value from the native code sub-routine to the proxy sub-routine.

25 (Canceled).

26. (Original) The apparatus of claim 19, further comprising an inter-process communication mechanism that is configured so that the proxy sub-routine and the native code sub-routine can communicate.

27. (Original) The apparatus of claim 19, further comprising an address width translating mechanism that is configured to translate an address from a first address width in the computer program to a second address width in the native code sub-routine.